CS229

Making Haptic Computer Vision a Bit Shifty: Downsizing Binary Image Classification Networks for the Edge

Philip M. Pfeffer
Department of Electrical Engineering
Stanford University
philpfef@stanford.edu
CS229, CS230

Abstract

A new optimiser, Pow2Opt, is proposed for restricting neural network parameter values in a binary image classification context. The intention is to downsize networks for use on consumer edge devices, such as the Arduino Nano BLE Sense, while optimising accuracy, to build a haptic vest. 76.6% accuracy is achieved with a 514KB fully connected neural network.

1 Introduction

Imagine being visually impaired but still able to feel the movement of surrounding crowds and cars on your body. This can be achieved by wearing a 'haptic vest' that uses a camera to capture images, a computer vision model to process them and coin-sized vibration motors that vibrate the shirt to communicate the output (1). However, computer vision models tend to be large and require many matrix multiplications to perform inference on a new image. This poses challenges on consumer edge devices, such as the Arduino Nano BLE (2) Sense and Raspberry Pi 4 (3), where memory and compute are scarce.

This paper describes methods for downsizing computer vision models by introducing a new optimiser for parameter updates that restricts weights to powers of two: Pow2Opt. The input to our algorithm is a (96, 96) greyscale image. A fully connected neural network (NN) is then used to output a predicted classification of the image. The models classify the image into the classes ['person', 'car'], which represent classes a haptic vest may need to recognise. Unsuccessful optimisation methods are briefly described in the appendix.

This model is evaluated on two metrics: the optimising metric is accuracy, and the size satisficing metric, which demands that the model parameters be represented in <1MB. Compared to the benchmark retrained convolutional neural network (CNN) MobileNetV2 (4; 5), which achieves 98.5% accuracy and occupies 1.6MB, and the benchmark fully connected NN without using Pow2Opt, which achieves 86.7% accuracy in 2.8MB, the fully connected NN using Pow2Opt achieves 76.6% accuracy with a 514KB model. Although accuracy noticeably drops compared to the benchmarks, the Pow2Opt model fits within the size constraints smaller and has the added benefit that all weights are represented as fixed-point numbers, turning expensive floating point multiplications into cheap bit-shift operations.

This work is conducted alongside a CS230 paper by Philip Pfeffer and Raul Dagir (1), which uses a similar dataset for three-class classification and implements a MobileNetV2 model without parameter modifications and is used in this paper as an estimate for the upper bound of the accuracy we can expect from a smaller model. Code for both of these papers can be found at `https://github.com/PhilipPfeffer/haptic_vest`[1].

2 Related Work

There are many efforts to downsize models for edge devices by quantising parameters - applying functions to reduce their size (in bytes) - and writing custom optimisers that restrict the set of values parameters can take on.

2.1 Binarized Neural Networks (BNNs):

Introduced by Matthiew Courbariaux, Yoshua Bengio et al. in 2016 (6), BNNs attempt to improve the power efficiency of neural networks while maintaining near state-of-the-art by restricting weights and post-Batch Norm

[1]The main files implemeted for this paper are `data_pipeline.ipynb`, `image_preprocessing.ipynb`, `customnn.ipynb`, `CS229_mobilenet_retrain.ipynb` and `resnet34_oracle.ipynb`.

activations to either +1 or -1 using the $\text{Sign}(x)$ function. This reduces memory size requirements and allows the use of hardware-efficient bit-wise operations. During training time, since the sign function's derivative is always zero (except when undefined at $x = 0$), a $\frac{\partial C}{\partial x}$ must be redefined. The authors created a gradient estimator, $\frac{\partial C}{\partial x_{binarized}} = Clip(Update(x), -1, 1)$, where x represents the true gradient $\frac{\partial C}{\partial x}$, which is calculated by caching the un-binarized activations. This gradient estimator produces clipped, but non-binary gradients. To account for the non-binary update to weights, all weights are re-binarized during the forward propagation.

This is similar to this paper's approach of rounding weights to powers of two to allow for use of bit-shifts instead of matrix multiplications. However, as discussed in the results section, this method was ineffective for this dataset.

2.2 Ternary and Low Bitwidth Neural Networks

XOR-Net uses similar principles to BNNs, implementing multiply accumulates with binarised weights as XOR operations, which are efficiently executed in hardware (7). Ternary Weight Networks (TWNs) (8) also apply similar reasoning to BNNs, but differ in their restricting parameters to the set $\{-1, 0, +1\}$. TWNs claim to benefit from more expressivity in the network, while simplifying multiply-accumulate opertations, since terms multiplied by 0 need not be accumulated. Low bitwidth neural network DoReFa-Net (9) attempts to reduce the size of parameters in the network, though do not place as heavy constraints as the preceding three models. DoReFa-Net further restricts gradients to be 6-bit numbers at training time. This allows for faster training, which is not relevant to the scope of this paper.

2.3 Quantization for Integer-Arithmetic-Only Inference

Several papers attempt different integer quantization techniques. This Google-published paper (10) demonstrates that training weights and biases as 32-bit floating point numbers and then quantizing weights to 8-bit integers and biases to 32-bit integers only reduces the accuracy of ResNet on image net by 1-3%. While the model size is made $4\times$ smaller, inference is only sped up with the use of ARM NEON Intrinsics, which are not present on many consumer edge devices.

3 Dataset

The models in this paper are trained using images from two publicly-available datasets. The COCO 2017 (11) dataset is used to source 'person' images. COCO labels up to five objects of interest with their bounding boxes and pixel segmentations for 80 image classes. This pixel segmentation information allows for selective use of 'person' images, described in figure 1. Images of the 'car' class were originally sourced from COCO, but labelling was found to be inaccurate for the haptic vest application. Instead, the Stanford Cars dataset (12) is used to source 'car' images.

3.1 Regularisation & Preprocessing

All images are cropped and resized to be 96×96, converted from 3-channel RGB to 1-channel greyscale, regularised between [0,1] and converted from the `float32` datatype to the `uint8` datatype by `scikit-image` (13). These preprocessing transforms are applied with low-compute consumer edge devices in mind. 96×96 pixels was determined to be a reasonable compromise between a resolution where humans can still identify objects and the number of imput parameters is low (which affects the number of multiply-accumulate operations needed during inference). Further, this is a reasonable resolution for off-the-shelf electronics hobbyist cameras to produce, for example, the OV7670 CMOS VGA Camera Module (14). Using 1-channel greyscale images further decreases the number of multiply-accumulates needed in the first layer of the NNs.

3.2 Dataset 'Person' Images

COCO provides many images that contain people. However, not all of these should be considered 'person' for the haptic vest application. This arises because the haptic vest should only react (detect) a person if they are close to the user. Otherwise, the vest will react to objects not relevant to the user's immediate surroundings. As a proxy for proximity, a hyperparameter called `area_threshold_percentage` is implemented. This variable indicates the minimum percentage area of the image that an object must occupy for the image to be included as an example of that category, calculated using COCO pixel segmentation information. The threshold percentage was set to 5%.

COCO was initially used to source 'car' images. However, only 2,482 COCO images surpass the 5% area threshold and those images are freqently mislabelled for the haptic vest application. Instead, images from the

	Train	Dev	Test
Person	21,810	2,671	1,410
Car	12,107	2,670	1,411

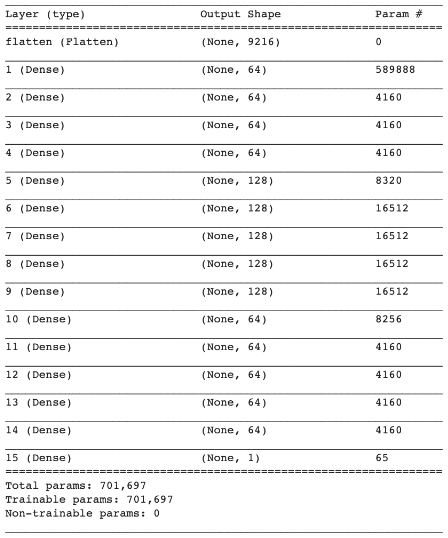
Figure 1: Haptic Vest binary classification dataset distribution using COCO and Stanford Cars.

Stanford Cars dataset are used to train the models. Stanford Cars images depict cars in the forefront of the image, whereas COCO images have no such guarantee. This allowed us to train, develop and test using images closer to the ground truth distribution of situations the haptic vest should react to a car.

4 Methods

4.1 Fully Connected Network

The input to our network is a 96×96 greyscale image. We then use a fully connected neural network (NN) to output a predicted classification of the image using a sigmoid output. The models classify the image into the classes ['person', 'car'], which represent classes a haptic vest may need to recognise. The architecture, shown in figure 2, flattens the input image into a single $96 \times 96 = 9,216$ element input array and propagates the input through only fully connected (Dense) layers. During inference, each node in the fully connected layer calculates the dot product of its inputs with its weight matrix, and then adds its biases: $Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$, where l represents the network's l^{th} layer. The network then passes $Z^{[l]}$ through a non-linear ReLU activation function, $A^{[l]} = max(0, Z^{[l]})$ to compute that layer's activation. The dot product normally requires many floating point multiplications to compute, but is turned into bit shift operations by the Pow2Opt optimiser proposed in this paper. The final layer of the model uses a sigmoid activation function, $\sigma(Z^{[l]}) = \frac{1}{1+exp(-(W^{[l]}X^{[l]}+b^{[l]}))}$, which outputs a single number, a score between 0 and 1. A score of less than 0.5 classifies the image as a 'person', otherwise the image is classified as a 'car'. The model was created using `keras` (15), which is built on `TensorFlow` (16).



```
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
1 (Dense)                    (None, 64)                589888
_________________________________________________________________
2 (Dense)                    (None, 64)                4160
_________________________________________________________________
3 (Dense)                    (None, 64)                4160
_________________________________________________________________
4 (Dense)                    (None, 64)                4160
_________________________________________________________________
5 (Dense)                    (None, 128)               8320
_________________________________________________________________
6 (Dense)                    (None, 128)               16512
_________________________________________________________________
7 (Dense)                    (None, 128)               16512
_________________________________________________________________
8 (Dense)                    (None, 128)               16512
_________________________________________________________________
9 (Dense)                    (None, 128)               16512
_________________________________________________________________
10 (Dense)                   (None, 64)                8256
_________________________________________________________________
11 (Dense)                   (None, 64)                4160
_________________________________________________________________
12 (Dense)                   (None, 64)                4160
_________________________________________________________________
13 (Dense)                   (None, 64)                4160
_________________________________________________________________
14 (Dense)                   (None, 64)                4160
_________________________________________________________________
15 (Dense)                   (None, 1)                 65
=================================================================
Total params: 701,697
Trainable params: 701,697
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Custom fully connected neural network with 701,697 parameters.

4.2 Loss function

Since this project explores the binary classification problem, the model uses the binary cross-entropy loss,

$$\mathcal{L} = \sum_{i=1}^{m} \left(y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right).$$

4.3 Pow2Opt Optimiser

The Pow2Opt optimiser restricts the parameters of the network to be positive or negative powers of two. Specifically, all weights are of the form $\pm 2^p$, where $p \in \mathbb{Z}$, meaning that $-2^3 = -8$ and $2^{-2} = 0.25$ are valid parameter values. Restricting the parameters is achieved by modifying the traditional gradient descent update step:

$$W := W - \alpha \frac{\partial J}{\partial W} \qquad b := b - \alpha \frac{\partial J}{\partial b},$$

where J represents the cost function and α represents the learning rate, to be:

$$var = \left\{ W - \alpha \frac{\partial J}{\partial W}; b - \alpha \frac{\partial J}{\partial b} \right\}$$

$$var := \text{Sign}(var) \cdot 2^{\text{round}(\log_2 \text{abs}(var))}.$$

This simply rounds the updated parameters, W and b, to the nearest power of 2, while preserving their signs.

Decimal	Sign	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}
-0.03125	1	0	0	0	1	0	0	0	0
0.375	0	1	1	0	0	0	0	0	0

Figure 3: Example of signed fixed point binary representation

To restrict the parameters to the $\pm 2^p$ form Pow2Opt requires, parameters can be represented in the fixed-point format, which allows for powers of two with negative exponents, as well as the usual powers of two with positive exponents. An example of the scheme is given in figure 3. Since Pow2Opt restricts weights to be strict powers of two, the fixed point representation of these numbers will be one-hot (excluding the sign bit). This means that parameters values can be further compressed from fixed-point numbers into a representation with two parts: one bit that indicates the sign, and n bits that store the index of the 1 in the one-hot fixed point representation. For example, using the example for -0.03125 above, you would store '1011', where the first '1' bit represents the sign, and '011' represents that the third index (using zero-indexing) in the fixed point representation, 2^{-5}, should be a '1'.

Ensuring parameters fit the form $\pm 2^p$ means that, during inference time, the inputs can be multiplied by the one-hot fixed point numbers by applying a bit shift rather than by performing a floating point multiplication. For example, if we set the fixed-decimal to be the after the least significant bit and ignore the sign bit for simplicity, then we obtain the following expression:

$$17_{10} \times 0.5_{10} = 100010_{2,\text{fixed-pt}} \times 000001_{2,\text{fixed-pt}} = 100010_{2,\text{fixed-pt}} >> 001_{2,\text{compressed}} = 010001_2 = 8.5_{10}.$$

Thus, during inference, the parameters trained using Pow2Opt can be multiplied with any fixed point number using only bit shifts.

Since this optimiser does not restrict the size of the exponent, p, parameters can become arbitrarily large or small during training. Thus, one cannot know how many bits are needed to describe the fixed point representation or the compressed representation before the model is trained. Therefore, model size implications are discussed in the results section, where we describe the range of values the network parameters takes on.

5 Results

This model is evaluated on two metrics: the optimising metric is accuracy, and the size satisficing metric, which demands that the model parameters be represented in <1MB. The fully connected NN using Pow2Opt achieves 76.6% accuracy on the test set with a model size of 514 KB using the compressed fixed point representation described in the Methods section. This paper compares its performance against a MobileNetV2 (4) convolutional neural network (CNN) architecture, which achieves 98.5% on the test set in 1.6MB, and against the same fully connected NN using Adam (17) as its optimiser, which achieves 86.7% accuracy in 2.8MB. Neither benchmark has restrictions applied to its parameters.

5.1 Model Size

After training, the model's maximum and minimum parameters (ignoring their signs) are `2.3841858e-07` $= 2^{-22}$ and `0.25` $= 2^{-2}$. This means that every parameter in the network fits the fits the form $\pm 2^p, p \in [-22, -2]$, so, excluding the sign, the model can only take 21 different values (42 total). Therefore, using the compression scheme discussed in the Pow2Opt subsection of Results, every parameter can be compressed to 1 sign bit + $\lceil \log_2(21) \rceil$ value bits $=$ 6 bits. Assuming that the model is stored in one contiguous binary array, this model takes 6 bits $\times$ $701,697$ parameters $\times$ $\frac{1}{8 \times 1024}$ bits per kilobyte $=$ 514 kilobytes.

5.2 Pow2Opt

Training the fully connected NN on the binary classification using Pow2Opt as the optimiser, a learning rate of 0.01 was experimentally derived. A batch size of 32 was used to train using a Google Colaboratory GPU. An overall accuracy on the of 76.6% was achieved on the test set, producing the confusion matrix in figure 4. Figure 5 shows the nine highest loss images in the test dataset. Since these are all car images, the nine images with the highest losses excluding car are shown in figure 6. We see that incorrect car images do not appear 'person'-like and vice-versa. This may be because due to the model architecture. Unlike this NN, CNNs maintain an explicit hierarchical representation of features, and different parts of the image share parameters. The NN, on the other hand, may not be able to take advantage of identifying similar features in different parts of the image. Therefore, it may be less able to pick up low-level features, such as the sharp edges of cars clearly visible in figure 5.

The drop in accuracy of the Pow2Opt model compared to the fully connected and MobileNetV2 benchmarks may be explained by the small set of values that Pow2Opt allows parameters to take on. Restricting the parameters of the network to quantised steps restricts the range of activations the network is able to represent. Therefore, it reduces the expressivity of the network.

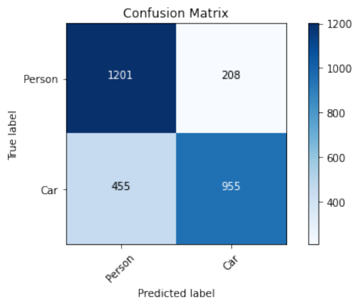


Figure 4: Test Set Confusion matrix for Pow2Opt Neural Network

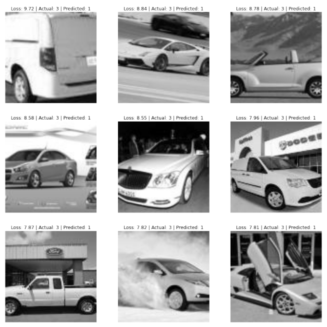


Figure 5: Nine highest loss images for Pow2Opt Neural Network

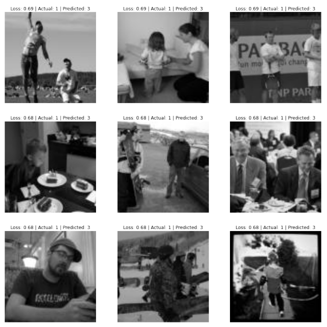


Figure 6: Nine highest loss, excluding car images

The learning plots for accuracy and loss, figure , show that the model is unstable at the beginning and eventually starts to stabilise. This change in stability may arise because the network is initialised with Xavier initialisation (18), meaning that parameters will start larger. The difference between higher powers of 2, e.g. between 0.5 and 0.25, are larger than lower powers of two, e.g. 0.125 and 0.0625. Therefore, initial updates to parameters may be more volatile as the network is not able to directly modify parameters to their ideal traditional gradient-descent derived values. Instead, the network must train other parameters to offset this effect, which takes more epochs to achieve.

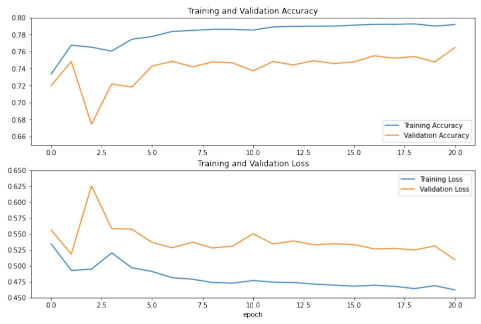


Figure 7: Learning Plot for Pow2Opt Neural Network

5.3 Unsuccesful Optimisers

Two optimisers attempted are described here along with analysis of their failure modes.

Initially, a similar update equation to Pow2Opt was designed that forces parameters to be integers, IntPow2Opt (positive exponent powers of two). IntPow2Opt causes loss to balloon during training, either rapidly becoming a 'nan' (due to integer overflow) or converging slowly from an unreasonably large loss, depending on the value of the learning rate. These observations persisted even if the parameters were clipped at [-32, 32] or [-4,4]. Furthermore, with clipping, most parameters took on extreme values or values close to zero. This suggests that the optimiser is susceptible to vanishing and exploding gradient problems. Predictions from the network trained with this optimiser are no better than random.

An optimiser mimicking the ternary neural network was also implemented, TernOpt. This optimiser forces network weights to be in the set $\{-1, 0, +1\}$. Unlike IntPow2Opt, the loss does not balloon. However, no learning rate or weight initialisation scheme was able to improve predictions beyond random guessing. The hypothesis for these observations is that using such a restrictive set of parameters, combined with the small size of the fully connected network, meant that the parameters are not expressive enough to fit the dataset.

6 Conclusions & Future Work

The proposed Pow2Opt optimiser achieves 76.6% accuracy on a fully connected NN with a model size of 514 KB. The model size reduction satisfies the 1MB limit at the cost of accuracy. Therefore, two future experiments of interest are to increase the size of the fully connected network and to apply the optimiser to different CNN architectures. Since the model presented only occupies 514KB, it would be interesting to add more parameters to the fully connected NN to understand if it improves accuracy. Further, changing the fully connected architecture to a CNN and applying Pow2Opt may drastically improve results and decrease the number of parameters needed. The latter was attempted, but more fine-tuning is needed to publish the results. I extend my thanks to the CS229 course staff, in particular to my TA advisor, Daniel Do.

7 Contributions

This work was entirely developed by Philip Mateo Pfeffer, including the retrained MobileNetV2 model developed primarily for use in an adjacent CS230 project (1). That project is co-authored by Philip Mateo Pfeffer and Raul Gallo Dagir (rgdagir@stanford.edu). The CS230 project takes advantage of work done to develop this paper's binary classification dataset for its three-class classification problem. Further, the CS230 paper is informed by results from the ResNet34 (19) oracle developed in earlier drafts of this paper.

References

[1] P. Pfeffer and R. Dagir, "Haptic vest: Image classification on the edge," 2020.

[2] "Arduino nano ble sense." [Online]. Available: https://store.arduino.cc/arduino-nano-33-ble-sense

[3] "Raspberry pi 4 model b: 4gb ram." [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/?resellerType=home

[4] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[5] J. Su, J. Faraone, J. Liu, Y. Zhao, D. B. Thomas, P. H. W. Leong, and P. Y. K. Cheung, "Redundancy-reduced mobilenet acceleration on reconfigurable logic for imagenet classification," Cham, pp. 16–28, 2018.

[6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016. [Online]. Available: https://arxiv.org/abs/1602.02830

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," 2016.

[8] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016.

[9] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2018.

[10] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017.

[11] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[12] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[13] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: image processing in Python," *PeerJ*, vol. 2, p. e453, 6 2014. [Online]. Available: https://doi.org/10.7717/peerj.453

[14] OmniVision, "Ov7670 cmos vga camera module." [Online]. Available: http://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf

[15] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[18] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[20] H. Chen and C. Su, “An enhanced hybrid mobilenet,” pp. 308–312, 2018.